Vysoká škola báňská – Technická univerzita Ostrava

Fakulta elektrotechniky a informatiky

Katedra informatiky

Absolvování individuální praxe

Individual professional practice in the company

2014

Vladimír Kantoš

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Vladimír Kantoš**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: KVADOS, a.s.
2. Struktura závěrečné zprávy:
a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta
b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti
c) Zvolený postup řešení zadaných úkolů
d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe
e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe
f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

Seznam doporučené odborné literatury:

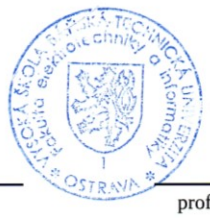
Podle pokynů konzultanta, který vedl odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Mgr. Marek Menšík, Ph.D.**

Konzultant bakalářské práce: Ing. Antonín Vaněček

Datum zadání: 01.09.2013
Datum odevzdání: 07.05.2014



doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: 22. 6. 2014

.....
podpis studenta

Poděkování

Rád bych poděkoval Ing. Jiřímu Vidlářovi za umožnění absolvování praxe. Rád bych také poděkoval Mgr. Marku Menšíkovi Ph.D. a týmu Ing. Antonína Vaněčka za odbornou pomoc, konzultaci a cenné rady při vytváření této bakalářské práce.

Prohlášení zástupce spolupracující právnické nebo fyzické osoby

„Souhlasím se zveřejněním této bakalářské/diplomové práce dle požadavků čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v bakalářských/magisterských programech VŠB-TU Ostrava."

Dne: 22. 6. 2014

.....
podpis zástupce

Abstrakt

Tato práce popisuje průběh mé odborné praxe ve firmě KVADOS, a. s. Bakalářskou odbornou praxi jsem vykonával od 3. října 2013 do 18. dubna 2014. V první části se zaměřuji na informace o firmě, její působení a zaměření. Dále popisuji úkol, na kterém jsem pracoval po dobu vykonávání praxe. Ke konci této práce jsem popsal znalosti, které jsem získal při studiu na VŠB a uplatnil je při praxi, znalosti nově nabyté při praxi a také hodnotím přínos celé praxe.

Klíčová slova

KVADOS; a.s., Odborná praxe; .NET; C#; Xamarin; Android; Windows Phone; iOS; WebAPI; OWIN; Azure; Multiplatformní mobilní klient; Vícevrstvá architektura; Modulární aplikace

Abstract

This thesis describes the process of my professional practice in the company KVADOS, a. s. I was performing bachelor's professional practice from 3rd October 2013 to 18th April 2014. In the first part I'm focusing on information about company and its operation. Furthermore I mention the task that I've been involved in during professional practice. By the end of this paper, I described the knowledge that I gained while studying at the VŠB, and which I applied in practice, knowledge acquired in practice and evaluated the contribution of the whole professional practice.

Key words

KVADOS; a.s., Professional practice; .NET; C#; Xamarin; Android; Windows Phone; iOS; WebAPI; OWIN; Azure; Multiplatform mobile client; Multilayer architecture; Modular application

Seznam použitých zkratek

ZKRATKA	VÝZNAM
WCF	Windows Communication Foundation
API	Application Programming Interface
OWIN	Open Web Interface for .NET
IIS	Internet Information Services
ODATA	Open Data protocol
DTO	Data Transfer Object
EF	Entity Framework
SCSF	Smart Client Software Factory
MVVM	Model-View-ViewModel
IOC	Inversion of Control

Obsah

1 Úvod

Praktická aplikace získaných poznatků je důležitým prvkem vzdělání. Z tohoto důvodu jsem se před dvěma roky rozhodl pracovat při studiu. Získal jsem stáž ve společnosti KVADOS, a.s., kde jsem se stal členem týmu v oddělení výzkumu a vývoje. Později jsem se rozhodl využít spolupráce mezi společností KVADOS, a.s., a VŠB-TU Ostrava a absolvovat zde i odbornou praxi jako svou bakalářskou práci.

Náplň odborné praxe byla zvolena tak, aby odpovídala strategickým požadavkům společnosti i náplni práce týmu Ing. Antonína Vaněčka. Tento tým se v současnosti skládá z deseti programátorů. Náplní týmu je především vývoj interního aplikačního frameworku pro serverové, desktopové a mobilní aplikace na platformě .NET a Android. Zadaný úkol jsem však vypracovával mimo tento framework.

1.1 O firmě

Společnost KVADOS, a.s., se zabývá vývojem vlastních softwarových řešení. Společnost působí v Ostravě již od roku 1992 a zaměřuje se především na klienty ze segmentu obchodu a služeb. Těmto klientům poskytuje software, jehož cílem je pomoci zlepšit kvalitu řízení procesů. Působnost společnosti je na českém i zahraničním trhu.

KVADOS, a.s., je držitelem několika certifikátů mezinárodních standardů ISO a je také členem IT Clusteru stejně jako Fakulta elektrotechniky a informatiky VŠB-TU Ostrava.

2 Zadané pracovní úkoly

Náplň odborné praxe tvořila práce na jednom komplexním úkolu. Úkolem bylo vytvořit proof-of-concept pro vývoj mobilního aplikačního frameworku s dosažením co největšího sdílení kódu a ověřit možnosti napojení na serverovou aplikaci vystavující své služby pomocí ASP.NET Web API. Tento úkol pokryl celou časovou dotaci pro praxi.

2.1 Koncept řešení multiplatformního mobilního klienta s napojením na ASP.NET Web API

Stávající aplikační framework pro mobilní aplikace používaný ve společnosti je postaven na principech Smart Client Software Factory. Tento framework je implementován pro platformu Android a již zastaralou platformu Windows Mobile. Vzhledem k nynějším trendům se očekává, že zákazníci budou vyžadovat také jiné platformy, jako například Windows Phone nebo iOS.

Serverová část stávajícího frameworku využívá pro komunikaci s klienty technologie WCF (Windows Comunication Fundation). Technologie WCF je dostatečně flexibilní pro jakékoliv potřeby, ale náročná na přesnou konfiguraci. Z tohoto důvodu se součástí úkolu stalo rovněž otestování schopností relativně nové technologie ASP.NET Web API. Dalším dílčím úkolem bylo zjištění prerekvizit pro nasazení serverové aplikace jako služby v cloudu, konkrétně v rámci Microsoft Azure.

Mým úkolem tedy bylo vytvořit jednoduchou multiplatformní mobilní aplikaci, kde byl důraz kladen na co nejvyšší míru sdílení kódu pro úsporu nákladů na vývoj. V aplikaci jsou zahrnuty klíčové schopnosti aplikačního frameworku, tedy přístup k serveru pomocí http protokolu, práce s lokální databází a fotoaparátem, získání umístění zařízení, spolupráce s mapovou aplikací a odesílání souborů. Úkolem serverové strany je přijímat a poskytovat data a soubory z klientské aplikace.

2.1.1 Technologické požadavky na serverovou stranu

Hlavním požadavkem na technologií použitou na serverové straně aplikace je ASP.NET Web API jako technologie pro vystavování služeb. Vzhledem k nynějšímu řešení, které je hostované jako samostatná aplikace, bylo dalším technickým požadavkem využití rozhraní OWIN (Open Web Interface for .NET) pro hostování této aplikace mimo IIS (Internet Information Services). Pro zjednodušení komunikace mezi mobilní a serverovou aplikací bylo dohodnuto využití protokolu OData (Open Data) pro získávání a publikování dat. Dalším požadavkem byla možnost umístit tuto aplikaci jako cloud service v Microsoft Azure.

2.1.2 Technologické požadavky na klientskou stranu

Klientská mobilní aplikace je cílena na nejběžnější mobilní platformy, tedy Windows Phone, iOS a Android. Z tohoto důvodu bylo hlavním technologickým požadavkem na klientskou mobilní aplikaci využití nástroje, který sjednocuje vývoj pro všechny platformy. Vzhledem k širokému

využití platformy .NET a jazyka C# pro vývoj byl vybrán nástroj Xamarin. Tento nástroj umožňuje vytvářet aplikace pomocí standardních nástrojů pro .NET, tedy například Visual Studia 2013, pro vývoj aplikací na platformy Android a iOS, přičemž v obou případech využívá projektu Mono. Projekt Mono je open source implementace prostředí .NET pro operační systémy postavené na platformách UNIX, Linux, BSD a Windows.

2.2 Časová náročnost jednotlivých částí úkolu

Úkol je rozdělen na dvě samostatné části. První částí je vývoj serverové aplikace, druhou vývoj multiplatformní mobilní aplikace. Časová náročnost jednotlivých částí je vyobrazena v tabulce 1.

Tabulka 1: Časová náročnost jednotlivých částí úkolu

Úkol	**Počet člověkodnů**
Serverová aplikace	20
Klientské aplikace	40

3 Řešení zadaných úkolů

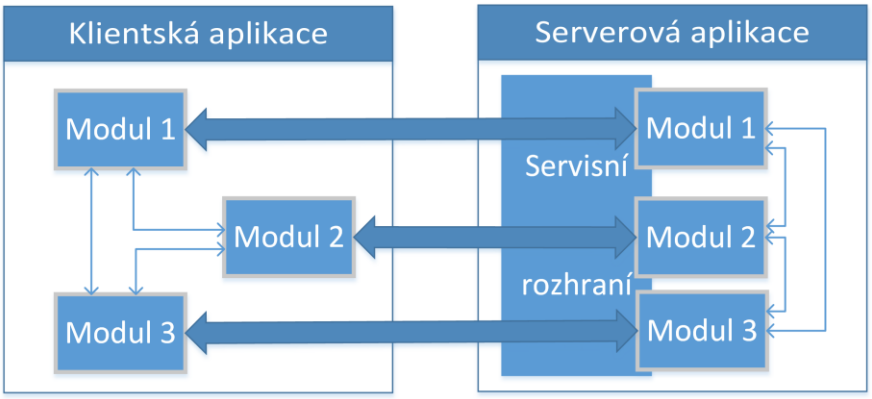
3.1 Architektura řešení

Koncept multiplatformní aplikace se drží již aplikovaných principů. Tyto principy patří k běžně využívaným principům při vývoji informačních systémů. Jejich hlavním cílem je usnadnit a urychlit vývoj a následnou údržbu daného systému.

3.1.1 Celková architektura řešení

Architektura aplikace částečně vychází ze stávajícího frameworku QAS. Základem je serverová část, která uchovává data a poskytuje je klientským aplikacím. Pro zjednodušení však nad těmito daty není na serverové straně prováděna žádná pokročilejší logika ani jiný způsob validace, než které poskytuje implementace OData protokolu a databázové úložiště. Klientská část aplikace pak tato data získává, zobrazuje a modifikuje na základě akcí uživatele. Na klientské straně nedochází k žádnému uchovávání dat, vše probíhá online oproti serverové databázi.

V rámci zefektivnění vývoje se ve stávajícím produktu využívá dělení aplikací do modulů. Jednotlivé moduly jsou projektovány tak, aby pokrývaly celou funkční oblast, jako například řízení lidských zdrojů, řízení vztahů se zákazníky a podobně. Tyto moduly existují jak na straně klientské aplikace, tak serverové části. Jednotlivé moduly mohou být závislé na jiných modulech, nikdy však na úrovni servisního rozhraní. Schéma komunikace je vyobrazeno na obrázku 1. Moduly mezi sebou komunikují vždy v rámci dané aplikace. Na servisní rozhraní serverového modulu odesílá dotazy vždy jen odpovídající modul klientské aplikace.

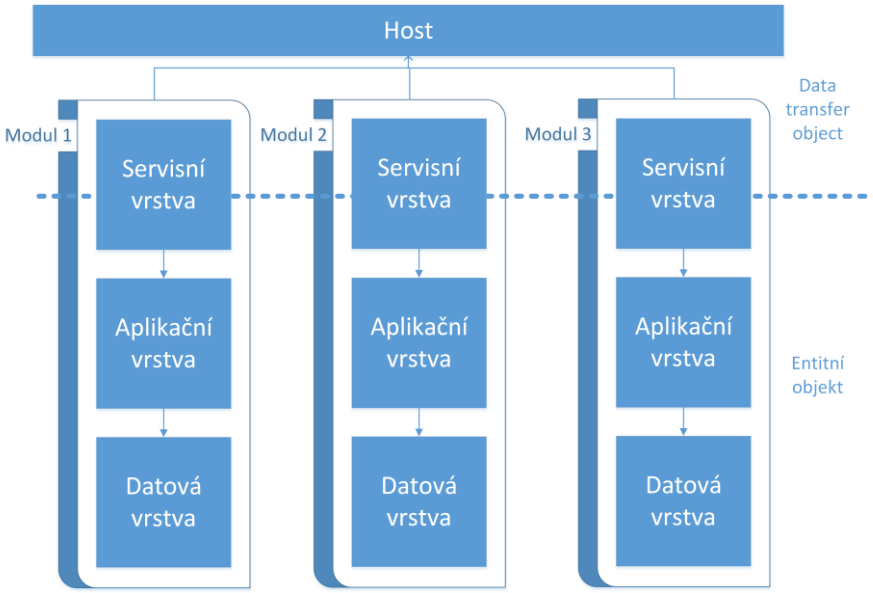


Obrázek 1: Schéma komunikace mezi moduly klientské a serverové aplikace

3.1.2 Architektura serverové části

Serverová část aplikace je tvořena z jednotlivých modulů, které pokrývají určitou funkční část aplikace. Tyto moduly jsou tvořeny pomocí vícevrstvé architektury. Nejvyšší vrstvou je servisní vrstva, která zajišťuje komunikační rozhraní pro klientské aplikace a také zajišťuje překlad

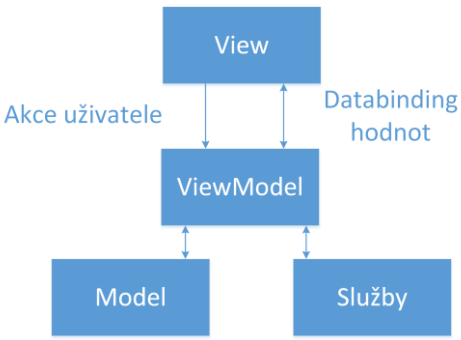
databázových entit na DTO (Data Transfer Object). Aplikační vrstva, která následuje, provádí veškerou logiku nad jednotlivými entitami. Pod touto vrstvou je datová vrstva, která slouží k uchovávání dat v databázi nebo souborovém úložišti. Datová vrstva je vytvořena s použitím návrhových vzorů Repository a Unit Of Work. Tyto moduly jsou zaregistrovány do mechanismu zajišťující hostování, který se stará také o ověřování uživatelů a zařízení.



Obrázek 2: Architektura serverové aplikace

3.1.3 Architektura klientské části

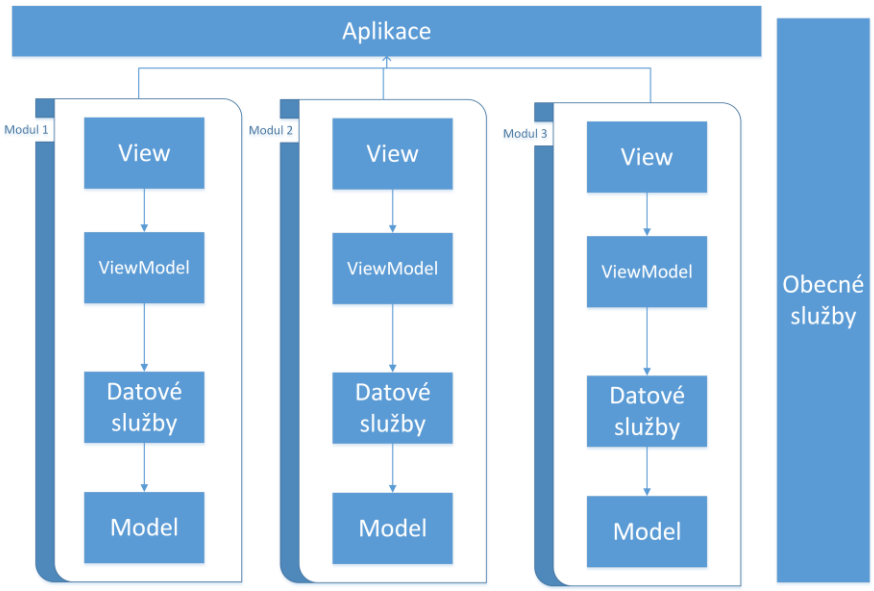
Klientská mobilní aplikace je postavena na architektonickém vzoru Model-View-ViewModel. Tento vzor odděluje uživatelské rozhraní (View) od bussiness logiky aplikace (Model), jejichž vazba je definována pomocí ViewModelu, který slouží jako konvertor hodnot, které jsou posléze zobrazovány v uživatelském rozhraní, a také zajišťuje prezentační logiku aplikace. Model klientské aplikace odpovídá serverovému DTO.



Obrázek 3: Návrhový vzor Model-View-ViewModel

Výhodou použití tohoto architektonického vzoru je v tomto případě také jednodušší vazba na části využívané v rámci stávající architektury SCSF, nad kterou je postaven aktuální framework a také vlastní nástroje urychlující vývoj.

Stejně jako serverová část aplikace je i klientská část rozdělena do modulů oddělujících jednotlivé funkční části. Jednotlivé moduly obsahují všechny prvky vzoru MVVM pro danou funkčnost, tedy jak reprezentaci uživatelského rozhraní, tak i prezentační a bussiness logiku. Dále tyto moduly mohou obsahovat obecné služby aplikace, které jsou dostupné nezávisle na funkčním kontextu, jako například služba pro otevření souborů mimo aplikaci.



Obrázek 4: Architektura klientské aplikace

3.2 Implementace řešení

Výše uvedená architektura je implementována na jednoduchém příkladu aplikace pro terénní pracovníky. Aplikace poskytuje možnost zobrazit úkoly přiřazené pracovníkovi, jejich umístění, přílohy a výkazy odvedené práce. Další schopností aplikace je zobrazit seznam zákazníků s jejich kontaktními osobami.

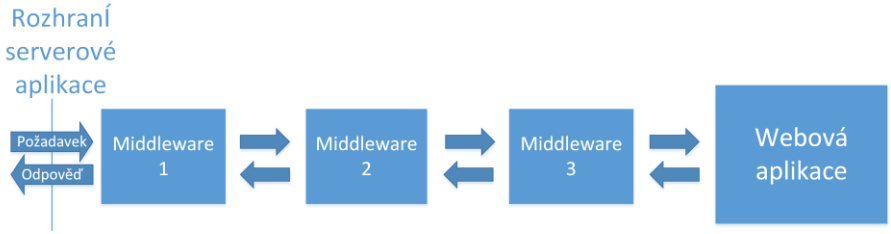
Obě části aplikace se skládají z hostovací aplikace a tří modulů. Tyto moduly se vztahují k jednotlivým oblastem funkčnosti aplikace. Prvním modulem je modul Core, který poskytuje základní infrastrukturu aplikace. Dalším modulem je modul CRM (Customer Relationship Management), který se stará o funkčnost týkající se zobrazování seznamu zákazníků. Poslední modul je nazván HRM (Human Resource Management). Tento modul zabezpečuje správu pracovních úkolů a výkazů práce.

3.2.1 Implementace serverové části

Základem této části je hostovací aplikace. Aplikace pro lokální hosting je vytvořena jako jednoduchá konzolová aplikace rozšířená o schopnost hostovat webové aplikace pomocí rozhraní OWIN. Implementaci tohoto rozhraní poskytuje NuGet balíček *Microsoft.Owin.SelfHost*, dříve známý pod označením Katana. Tato implementace využívá pro naslouchání na předdefinované adrese třídu *System.Net.HttpListener*. Tímto je docíleno naprostého oddělení od systému IIS. Při startu aplikace se vytvoří nová webová aplikace podle definovaného předpisu, která naslouchá na definované adrese.

Pro hostování v Microsoft Azure je využit obdobný přístup. Kromě nutného publikačního projektu (*QAS.Multiplatform.Demo.Server.Azure*) bylo potřeba vytvořit třídu, která reprezentuje danou službu, která je implementována ve třídě *QAS.Multiplatform.Demo.Server.Azure.Host.WorkerRole*. Tato služba se chová obdobně jako běžné Windows Services, tedy definuje metody *OnStart* a *OnStop*, ve kterých startujeme, respektive zastavujeme službu samotnou. Při startování se vytvoří webová aplikace stejným způsobem jako v případě lokálního hostování. Při zastavení služby je tato aplikace ukončena a odstraněna z paměti. Předpis pro aplikaci je stejný jako v případě lokálně hostované aplikace.

Předpis pro webovou aplikaci je definován ve třídě *QAS.Multiplatform.Demo.Server.Host.Startup*. Tato třída určuje všechny části, kterými projde příchozí požadavek na aplikaci a následující odpověď na tento požadavek. Jednotlivým vrstvám, které předcházejí aplikaci, se říká middleware. Tyto middleware jsou obdobou handlerů v IIS.



Obrázek 5: Schéma zpracování middleware v OWIN

Middleware umožňuje provést akce ještě před zpracováním požadavku v aplikaci, proto se využívají primárně pro validaci požadavku a získání dat z požadavku. Informace, které daný middleware z požadavku získá, se pak přenáší v kontextu, který je unikátní pro každý požadavek. Jednotlivé middleware je možné zřetězit a využívat tak informace získané předchozí vrstvou. V serverové části jsou využity middleware pro ověření uživatele pomocí uživatelského jména a hesla, pro ověření zařízení a vypnutí cache odpovědí na klientské straně.

Při startu aplikace je rovněž nutné zaregistrovat cesty, které je pak aplikace schopna obsloužit. Registrace se provádí v konfiguračních souborech modulů. V případě ASP.NET Web API jsou tyto cesty kompatibilní s konvencí ASP.NET MVC, kde je součástí cesty požadovaný Controller a metoda tohoto Controlleru. V případě použití OData protokolu je nutné zaregistrovat i tyto cesty. Pro tento krok je ovšem dostupná třída *ODataConventionModelBuilder* (součást NuGet balíčku *Microsoft.AspNet.WebApi.OData*), která pomocí reflexe vytvoří potřebná metadata OData rozhraní a zaregistruje potřebné cesty.

Základem aplikace samotné je IoC (Inversion of Control) kontejner. IoC umožňuje vytvořit volnější vazby mezi komponentami. Toho je dosaženo intenzivním využitím rozhraní tříd, proti kterému programujeme, namísto vytvoření konkrétní instance třídy implementující toto rozhraní. O poskytnutí této instance se stará práce IoC kontejner, který rozhodne o tom, jaká konkrétní implementace bude použita na základě registrace provedené při vytvoření tohoto kontejneru. V případě serverové části aplikace byla jako IoC kontejner využita knihovna AutoFac z důvodu integrace s OWIN implementací Katana a ASP.NET Web API. Tento kontejner rozhoduje také o použití správného ASP.NET Web API controlleru pro příchozí požadavek. Veškeré závislosti v aplikaci jsou řešeny pomocí IoC a jsou definovány v konstruktoru dané třídy. To vede k využití principu Constructor injection. Kontejner IoC při vytvoření objektu zjistí parametry konstruktoru pomocí reflexe a vytvoří odpovídající typy, které předá tomuto konstruktoru. Typy objektů v konstruktoru jsou ve většině případů rozhraní definující závislosti. Instance třídy je tedy vytvořena s již vyřešenými (resolved) závislostmi.

Na startu webové aplikace je rovněž provedena definice mapování mezi DTO a databázovými entitami. Pro toto mapování je využita knihovna Automapper. Tato knihovna usnadňuje mapování mezi typy a rovněž automaticky detekuje odpovídající mapování podle názvů vlastností jednotlivých tříd. Mapování lze definovat do profilů, které je pak možné podmíněně registrovat do samotného mapovacího enginu. Definice mapování umožňuje také ruční úpravu mapování včetně použití vlastních resolverů. Vlastní resolver byl použit při mapování ID položky v relaci na databázové entitě. Samotné třídy DTO jsou z důvodu sdílení s klientskou aplikací umístěny ve vlastním knihovním projektu, který je realizován jako PCL (Portable Class Library).

Použití modulu v aplikaci je dáno integrováním jeho konfigurace do aplikace. Konfigurace cest a IoC kontejneru je definována vždy v konfigurační třídě daného modulu. Jednotlivé konfigurace jsou definovány ve třídách *ModuleConfig* ve složce *App_Start* každého modulu. Zpracováním konfigurace modulu dojde k zaregistrování cest, mapování, OData entit, controllerů a tříd, které samotný modul tvoří. Modul sám o sobě je vytvořen jako samostatná ASP.NET Web API aplikace.

Pro samostatné spuštění modulu chybí jen samostatná konfigurace pro start aplikace. Modul kromě konfigurace obsahuje také třídy vrstev popsaných v architektuře.

Serverová aplikace obsahuje tři výše uvedené moduly – Core, CRM a HRM. Core je nejdůležitějším modulem, definuje totiž generické controllery, entitní služby, repositáře a UnitOfWork. Díky tomuto přístupu není nutné mít oddělené třídy pro jednotlivé entity, které sdílí identický kód, stačí pouze v IoC registrovat konkrétní kombinaci generické implementace a entity. Jedinou výjimkou jsou controllery, pro jejichž správné adresování v URL požadavku je nutné, aby existovala odpovídající třída. Tyto třídy ovšem dědí z bázového controlleru a definují odpovídající typy DTO a entit.

Controllery ASP.NET Web API jsou v této aplikaci využity jako koncové body pro konzumaci dat pomocí protokolu OData. Pro tuto funkci je nutné, aby controller dědil z *ODataController* (součást NuGet balíčku *Microsoft.AspNet.WebApi.OData*) a implementoval metody obsluhující HTTP metody Get, Put, Post, Patch, Merge a Delete. Metody Get je nutné anotovat atributem *Queryable*, aby bylo možné provádět nad výsledkem evaluaci OData Query. Při použití OData relací je nutné ještě vytvořit odpovídající metody pro získání navázaných dat. Controller se rovněž stará o mapování mezi entitním objektem a DTO. Po vytvoření entitního objektu z příchozího DTO je pomocí entitní služby proveden požadovaný úkon. Případný výsledek je namapován zpět na DTO a odeslán jako odpověď klientské aplikaci.

Entitní služby jsou, stejně jako controllery, vytvořeny genericky, vzhledem k jejich jednoduché funkci provádění CRUD (Create, Read, Update a Delete) operací nad datovým úložištěm. Implementací generické entitní služby je *QAS.Multiplatform.Demo.Server.Core.Services.EntityService*. Výjimkami jsou ověřovací služby (*LoginService*, *DeviceService*) a služby obsluhující souborové úložiště (*LocalFileStorageService* a *AzureFileStorageService*).

Datovou vrstvu tvoří implementace architektonického vzoru Repository a Unit of Work, reprezentované třídami *QAS.Multiplatform.Demo.Server.Core.Data.Repositories.Repository* a *QAS.Multiplatform.Demo.Server.Core.Data.UnitOfWork.UnitOfWork*, které se dále odkazují na jednotlivé datové kontexty. Tyto kontexty jsou vytvořeny pomocí Entity Framework Code First. Tento přístup byl využit z důvodu jednoduchosti a rychlosti vývoje. V průběhu vývoje bylo odzkoušeno použití technologie EF Code Migrations, avšak ta se posléze ukázala jako nevhodná pro tento účel. Jejím problémem byla neschopnost generovat správné migrace pro relace mimo daný kontext, využívané při odkazu na entitu z jiného modulu. Z tohoto důvodu je definice databázového modelu spolu se skripty pro vytvoření testovacích dat pro každý modul oddělena do separátních databázových projektů.

3.2.2 Implementace klientské části

Základem klientské aplikace je framework sjednocující přístup k vývoji aplikací na cílových platformách Windows Phone, Android a iOS. Jako sjednocující framework byl zvolen framework MVVMCross. Tento framework je open source a umožňuje tvořit aplikace pomocí principu MVVM,

který je zároveň nativním přístupem pro platformu Windows Phone. Oproti ostatním frameworkům podporujícím vývoj pomocí MVVM má výhodu právě v podpoře několika platforem. Nejbližším konkurentem je ReaktiveUI framework, který je postaven nad Reaktive Extension, ale komunita není tak široká jako u MVVMCross a také nesjednocuje schopnosti jednotlivých platforem (přístup k souborům, fotoaparátu atp.) v takové míře jako MVVMCross. Další výhodou MVVMCross je snadná rozšiřitelnost pomocí pluginů. MVVMCross je také využíván společností Microsoft pro vývoj aplikací platformy Nokia X Software Platform, vycházející z platformy Android.

Struktura aplikace vytvořené pomocí MVVMCross se skládá z několika projektů. Základem je PCL knihovna obsahující společný kód obsluhující bussiness logiku (Modely), datové služby a ViewModel. Dále je pak nutné vytvořit projekty aplikací, které referencují tuto knihovnu a přidávají Views, které jsou pro každou platformu specifické. V případě využití modulů je však nutné tento systém upravit. Každý modul tvoří dvě knihovny. Jádro modulu tvoří PCL knihovna tak jako v případě jednoduché aplikace, ale tato knihovna není přímo referencována aplikačním projektem. Druhou je knihovna specifická pro danou platformu, ve které jsou umístěny Views vztahující se k danému modulu. Tuto knihovnu pak referencuje projekt aplikace. Tímto oddělíme implementaci modulu od aplikace a je tedy možné snadněji zaměňovat jednotlivé moduly ve výsledné aplikaci. Jedinou výjimkou v tomto přístupu je platforma iOS, jejíž moduly jsou z důvodu nekompatibility knihoven umožňující využití async/await konstrukce tvořeny pouze jedním projektem. Tento projekt je specifický pro platformu a obsahuje třídy jádrové knihovny přidané jako odkazy. Změny kódu v PCL knihovně modulu se tedy projeví stejně jako u ostatních platforem. Kromě těchto projektů tvoří aplikaci ještě PCL knihovna obsahující DTO, která je sdílena se serverovou aplikací a knihovna obsahující další obecné nástroje suplující chybějící funkce mobilního prostředí.

Aplikační projekty jsou obdobou hostující aplikace v případě serveru. Předpis aplikace je definován ve třídě *Setup*. Třída *Setup* je specifická pro každou platformu a registruje assembly, které tvoří danou aplikaci. Na základě této registrace jsou pak vyhledávány ViewModely a jejich příslušná Views. V této třídě je také definován mechanizmus logování událostí a kořenový ViewModel, který definuje vstupní ViewModel a inicializuje IoC. IoC kontejner je implementován přímo v MVVMCross frameworku. Registrace do IoC je prováděna buď na základě konvence pojmenování tříd, nebo přímým definováním typu. Tyto typy jsou poté registrovány pod interface, který přímo implementují. Aplikační projekty také obsahují potřebné assety, dodatečné soubory, jako například konfigurační soubor. Tyto soubory jsou poté přibaleny do distribučního balíčku aplikace.

Jednotlivé moduly jsou do aplikace registrovány pomocí svých konfiguračních tříd. Konfigurační třídy jsou odděleny pro jádrové knihovny modulu a platformní knihovny. Toto rozdělení vzniklo z důvodu registrace jednotlivých knihoven a také možnosti odlišit způsoby registrace služeb bussiness logiky od služeb platforem.

Klientská aplikace obsahuje, stejně jako serverová aplikace, tři moduly – Core, CRM a HRM. V modulu Core jsou definovány všechny bázové třídy pro datové služby, View a ViewModely. Kromě těchto bázových tříd obsahuje také základní služby platforem pro práci se soubory, assety, zobrazení aktivity, dále pak službu pro přenos souborů, datovou službu pro číselníky, služby logování, konfigurace, správy uživatele a zařízení a také službu pro navigaci mezi

ViewModely. Moduly CRM a HRM obsahují datové služby a ViewModely pro odpovídající use case. Modul HRM obsahuje navíc službu pro zjištění aktuálního umístění zařízení.

Služba *AppConfigurationService*, starající se o konfiguraci aplikace, načítá konfiguraci z XML souboru. Prvotní konfigurace pochází z instalačního balíčku aplikace. Tato konfigurace je uložena do privátního úložiště aplikace a je do ní uložen i identifikátor dané instalace. Tím je možno odlišit jednotlivé zařízení.

Standardním přístupem k navigaci v MVVMCross je přímá navigace mezi ViewModely, které mají mezi sebou úzkou vazbu. Předávání komplexnějších parametrů není podporováno, je nutné tyto parametry serializovat. Pro uvolnění vazeb mezi ViewModely a usnadnění modularity je navigace prováděna pomocí *NavigationService*. Tato služba registruje jednotlivé ViewModely pod definovaným názvem a stará se o serializaci parametrů. Také umožňuje určit, které ViewModely jsou součástí aplikačního menu.

Pro sjednocení zobrazení akcí v nástrojových lištách aplikací byla vytvořena služba *BarService*. Tato služba je implementována každou platformou zvlášť. Jejím úkolem je zajistit zobrazení a odstranění tlačítek akcí z nástrojových lišt a také provádění příslušných akcí. Zdrojem dat této služby jsou samotné ViewModely, jejichž Commandy, které jsou určené k zobrazení v nástrojové liště, jsou anotovány atributem *BarCommand*. Tento atribut specifikuje ikonu a název akce a také platformy, na kterých se daná akce zobrazuje.

Datové služby v aplikaci jsou vytvořeny s pomocí knihovny *Simple.OData.Client*. Tato open source knihovna usnadňuje práci se službami vystavujícími data pomocí protokolu OData. Její hlavní výhodou je možnost fluentního zápisu OData dotazu a schopnost dynamicky se přizpůsobit rozhraní služby. Toho je dosaženo volnou vazbou mezi klientem a službou. Na rozdíl od přidání služby jako reference, je při prvním dotazu na službu proveden požadavek na metadata služby. Tato metadata pak slouží k sestavení samotného dotazu. Při změně rozhraní služby tedy není nutné upravit klienta služby, pouze datové objekty, které tento klient produkuje. Knihovna je intenzivně vyvíjena, z tohoto důvodu je v aplikaci referencován její lokální klon s malými opravami chyb, které se při vývoji projevily.

Knihovny modulů specifické pro jednotlivé platformy jsou tvořeny jednotlivými Views, konvertory hodnot a implementacemi platformních služeb. Názvy Views musí dodržovat určité jmenné konvence. Jednotlivé platformy si zachovávají původní přístupy k definici vzhledu, nahrazen je pouze programovací jazyk. U Windows Phone je tedy uživatelské rozhraní definováno pomocí stránky XAML (eXtensible MArkup Language). Pro svázání tohoto View s odpovídajícím ViewModelem je nutné třídu reprezentující toto View anotovat atributem *MvxPhoneView*. Views platformy Android jsou definovány obdobně jako v případě Windows Phone. I zde definuje View třída, která je odvozena od třídy Activity. Tato třída aplikuje vzhled popsaný pomocí jazyka XML. V případě iOS je vzhled definován pomocí .xib souboru, který je výstupem designeru vzhledu aplikace XCode. Tento popis je aplikován ve ViewControlleru, který definuje provázání s ViewModelem. Databinding je v případě Windows Phone a Androidu definován v popisu vzhledu aplikace, u iOS je nutné jej vytvořit v kódu ViewControlleru.

Zobrazování jednotlivých View je řízeno pomocí ViewPresenteru. V případě Windows Phone a Androidu je použit výchozí presenter MVVMCross. U iOS bylo součástí požadavků stálé zobrazení aplikačního menu. Pro tento účel bylo nutné vytvořit vlastní kořenový ViewController a vlastní presenter. Presenter pro zobrazení dvou ViewModelů vedle sebe je implementován ve třídě *QAS.Multiplatform.Demo.App.iOS.SplitPresenter*. Tento presenter je pak zaregistrován v *Setup* třídě aplikace platformy iOS. Řízení umístění zobrazovaného View je řízeno pomocí PresentationType atributu, kterým je každé View anotováno.

3.3 Zhodnocení zkušeností s vývojem

Toto hodnocení se zaměřuje především na nástroj Xamarin. Práce s ASP.NET Web API odpovídala očekáváním plynoucím z přímé integrace do používaného vývojového prostředí, kterým bylo Visual Studio 2013. Dokumentace ASP.NET Web API odpovídala standardům firmy Microsoft.

Nástroj Xamarin vznikl na základě existujícího projektu Mono. Tento projekt byl již delší dobu využíván, proto nebyl problém s využitím prostředí .NET na mobilních zařízeních. Problémy vyvstaly až s použitím dalších knihoven, které nebyly vždy kompatibilní s prostředím Xamarin, především ve verzi pro iOS. Tyto problémy se vyskytovaly primárně u BCL (Base Class Library) knihoven Microsoft.Net.Http a Microsoft.Bcl.Async při jejich použití v PCL. Dočasným řešením je přidání kódu těchto PCL do knihoven specifických pro iOS.

Vývoj samotný však neprovází přílišné obtíže. Míra sdílení kódu je závislá především na složitosti bussiness logiky a ViewModelů. V případě tohoto konceptu nebyla složitost velká, ale při pohledu na tabulku zjistíme, že se pohybuje okolo 50% - 60% v případě modulů CRM a HRM. Modul Core je vzhledem k bázovým implementacím náročnější na platformě specifický kód. Při dlouhodobějším vývoji lze však pomocí optimalizací a vhodně navrženého základního modulu dosáhnout vyšší míry sdílení.

Tabulka 2:Počet řádků kódu v jednotlivých knihovnách

Knihovna	***Počet řádků***
QAS.Multiplatform.Demo.App.Core	713
QAS.Multiplatform.Demo.App.Core.Droid	257
QAS.Multiplatform.Demo.App.Core.iOS	485
QAS.Multiplatform.Demo.App.Core.WP8	257
QAS.Multiplatform.Demo.App.Module.CRM.Core	215
QAS.Multiplatform.Demo.App.Module.CRM.Droid	39
QAS.Multiplatform.Demo.App.Module.CRM.iOS	113
QAS.Multiplatform.Demo.App.Module.CRM.WP8	49
QAS.Multiplatform.Demo.App.Module.HRM.Core	911
QAS.Multiplatform.Demo.App.Module.HRM.Droid	164
QAS.Multiplatform.Demo.App.Module.HRM.iOS	364
QAS.Multiplatform.Demo.App.Module.HRM.WP8	228

Xamarin je distribuován s vlastním vývojovým nástrojem Xamarin Studio. Jde o upravený klon open source nástroje SharpDevelop. Toto vývojové prostředí však nedisponuje tak pokročilými nástroji jako Visual Studio. Samotná integrace do Visual Studia však není bezchybná. Kvůli hloubce integrace dochází ke kolizím s jinými rozšířeními Visual Studia, například velmi oblíbeným rozšířením ReSharper firmy JetBrains. Integrace do Visual Studia se však stále zlepšuje jak kvalitativně, tak schopnostmi. V nově uvedené verzi 3 disponuje Visual Studio možností návrhu uživatelského rozhraní jak pro platformu Android, tak i iOS.

Využití Visual Studia pro vývoj na iOS s sebou však nese jisté břímě v nutnosti spojení s prostředím Xamarin nainstalovaném na počítači Mac se systémem OS X, který je označován jako Build Host. Spojení lze provést pouze mezi jedním Build Hostem a jednou instancí Visual Studia. Toto omezení znamená, že každý vývojář potřebuje vlastní počítač Mac pro vývoj na platformě iOS. Testovací mobilní zařízení nejsou vázaná na Build Host nebo vývojáře. Pro testování na zařízení je ovšem potřeba aktivní iOS Developer Subscription. Počáteční investice do vývoje pro platformu iOS tedy může být poměrně náročná

Celkový pohled na zkušenosti s vývojem multiplatformních aplikací shrnuje následující tabulka.

Tabulka 3: SWOT analýza multiplatformního vývoje pomocí Xamarin

Silné stránky	**Slabé stránky**
■ Vývoj pomocí jednoho jazyka ■ Známé vývojové prostředí ■ Sdílení stávajícího kódu ■ Rozrůstající se komunita ■ Okamžitá podpora nových verzí mobilních OS	■ Nutnost OS X pro vývoj na iOS ■ Nesjednocené UI (řeší Xamarin.Forms) ■ Horší integrace do VS
Hrozby	**Příležitosti**
■ Nekompatibilní knihovny třetích stran	■ Využití MVVM jako implementace WorkItemů SCSF

4 Chybějící znalosti

Na začátku práce jsem postrádal hlubší znalosti aplikačního rozhraní jednotlivých mobilních platforem. Tyto znalosti jsou částečně součástí předmětu Tvorba aplikací pro mobilní zařízení, ale tento předmět jsem neabsolvoval. Obtíže představovala také práce s asynchronním rozhraním v jazyce C#.

5 Znalosti a dovednosti získané během studia a uplatněné v průběhu odborné praxe

V průběhu praxe jsem využil především znalosti získané v předmětu Vývoj informačních systémů, zejména znalosti týkající se architektury aplikací. Dalším důležitým zdrojem znalostí byl předmět Architektura technologie .NET, kde jsem se naučil využívat pokročilejší funkce, které platforma .NET a programovací jazyk C# nabízí.

Závěr

Vytvořený koncept jsem na závěr své odborné praxe předal vedoucímu týmu k další analýze. Absolvování této odborné praxe mi přineslo nové znalosti jak v praktickém použití programovacího jazyka C# a schopností .NET, tak i potřebné zkušenosti s návrhem a vývojem architektonicky složitějších aplikací. Implementované vzory a principy jsou obecně aplikovatelné v mnoha řešeních, ať už při řešení školních projektů, nebo v praxi. Praktická zkušenost z firmy zabývající se vývojem softwaru je také neocenitelnou zkušeností.

Použitá literatura

[1] Xamarin Cross-platform Application Development. Birmingham: Packt Publishing, 2014. ISBN 978-184-9698-474.

[2] *Xamarin Mobile Application Development for iOS*. Birmingham: Packt Publishing, 2013. ISBN 978-178-3559-190.

[3] SZOSTAK, Tomasz. *Windows Phone 8 Application Development Essentials*. Online-Ausg. Birmingham: Packt Publishing, 2013. ISBN 978-184-9696-777.

[4] GAYLORD, Jason N, Christian WENZ, Pranav RASTOGI, Todd MIRANDA a Scott HANSELMAN. *Professional ASP.NET 4.5 in C# and VB*. l, 1389 pages.

[5] Home. *MvvmCross Wiki*. [online]. 27.4.2014 [cit. 2014-06-25]. Dostupné z: https://github.com/MvvmCross/MvvmCross/wiki

Příloha A

Obsah přiloženého CD

- Elektronická verze Bakalářské práce
- Zdrojový kód aplikace ve složce